Práctica de Organización del Computador II

Interrupciones

Segundo Cuatrimestre 2024

Organización del Computador II
DC - UBA

Introducción

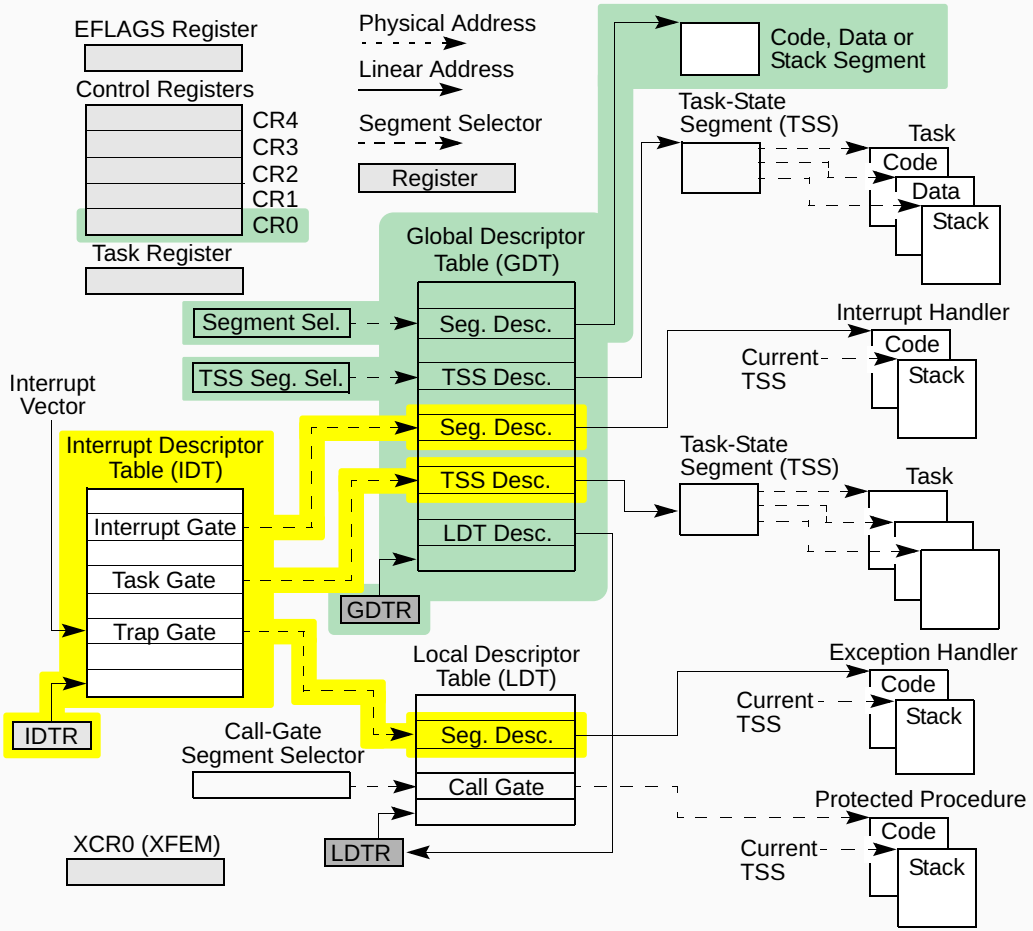
Sobre la clase de hoy

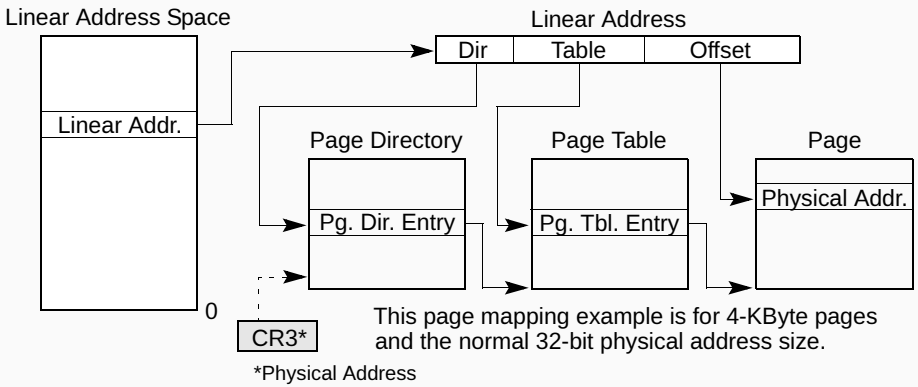
En la clase de hoy vamos a ver:

- **Repaso de interrupciones**
- **Estructura de la IDT**
- **Interrupciones de teclado y reloj**
- **Syscalls con interrupciones de software**

La idea es presentar la información necesaria para ayudarles a completar el taller.

Usted ...


EFLAGS Register
Control Registers
CR4
CR3
CR2
CR1
CR0
Task Register
Physical Address
Linear Address
Segment Selector
Register
Code, Data or Stack Segment
Task-State Segment (TSS)
Task
Code
Data
Stack
Global Descriptor Table (GDT)
Segment Sel.
Seg. Desc.
TSS Seg. Sel.
TSS Desc.
Seg. Desc.
TSS Desc.
LDT Desc.
GDTR
Interrupt Handler
Code
Current TSS
Stack
Interrupt Vector
Interrupt Descriptor Table (IDT)
Interrupt Gate
Task Gate
Trap Gate
Task-State Segment (TSS)
Task
Local Descriptor Table (LDT)
Exception Handler
Code
Current TSS
Stack
IDTR
Call-Gate Segment Selector
Seg. Desc.
Call Gate
Protected Procedure
Code
Current TSS
Stack
XCR0 (XFEM)
LDTR


Linear Address Space
Linear Addr.
0
Linear Address
Dir
Table
Offset
Page Directory
Pg. Dir. Entry
Page Table
Pg. Tbl. Entry
Page
Physical Addr.
CR3*
This page mapping example is for 4-KByte pages and the normal 32-bit physical address size.
*Physical Address

Repaso de interrupciones

Interrupciones

¿Qué son las interrupciones y para que las usamos?

- Conceptualmente permite a un agente externo o interno solicitar la interrupción de la ejecución actual para atender un pedido.
- La parte solicitante ve al procesador como un recurso al cual quiere tener acceso.

Interrupciones

¿Qué son las interrupciones y para que las usamos?

- El mecanismo implementado **define una identidad numérica para cada interrupción** y utiliza una tabla de descriptores donde para cada índice, o identidad, se decide:
  - Dónde se encuentra la rutina que lo atiende (dirección de memoria).
  - En qué contexto se va a ejecutar (segmento y nivel de privilegio).
  - De qué tipo de interrupción se trata.

Tipos de Interrupciones

¿Qué tipos de interrupciones vamos a usar en la práctica?

- **Excepciones** que van a ser generadas por el procesador cuando se cumpla una condición, por ejemplo si se quiere acceder a una dirección de memoria a través de un selector cuyo segmento tiene el bit `P` apagado.
- **Interrupciones:**
  - **Externas**, cuyo origen se da en un dispositivo externo, en nuestro caso van a ser **el reloj y el teclado**.
  - **Internas**, cuyo origen se da en una llamada a la instrucción `INT` por parte de un proceso.

Tipos de Excepciones

- **Fault:** Excepción que podría corregirse para que el programa continúe su ejecución. El procesador guarda en la pila la dirección de la instrucción que produjo la falla. Algunos *faults* suman un código de error a la pila.

- **Traps:** Excepción producida al terminar la ejecución de una instrucción de trap. El procesador guarda en la pila la dirección de la instrucción a ejecutarse luego de la que causó el trap.

- **Aborts:** Excepción que no siempre puede determinar la instrucción que la causa, ni permite recuperar la ejecución de la tarea que la causó. Reporta errores severos de hardware o inconsistencias en tablas del sistema.

Tipos de Interrupciones

Table 6-1. Protected-Mode Exceptions and Interrupts

Vector	Mnemonic	Description	Type	Error Code	Source
0	#DE	Divide Error	Fault	No	DIV and IDIV instructions.
1	#DB	Debug Exception	Fault/ Trap	No	Instruction, data, and I/O breakpoints; single-step; and others.
2	—	NMI Interrupt	Interrupt	No	Nonmaskable external interrupt.
3	#BP	Breakpoint	Trap	No	INT3 instruction.
4	#OF	Overflow	Trap	No	INTO instruction.
5	#BR	BOUND Range Exceeded	Fault	No	BOUND instruction.
6	#UD	Invalid Opcode (Undefined Opcode)	Fault	No	UD instruction or reserved opcode.
7	#NM	Device Not Available (No Math Coprocessor)	Fault	No	Floating-point or WAIT/FWAIT instruction.
8	#DF	Double Fault	Abort	Yes (zero)	Any instruction that can generate an exception, an NMI, or an INTR.
9		Coprocessor Segment Overrun (reserved)	Fault	No	Floating-point instruction.[1]
10	#TS	Invalid TSS	Fault	Yes	Task switch or TSS access.
11	#NP	Segment Not Present	Fault	Yes	Loading segment registers or accessing system segments.
12	#SS	Stack-Segment Fault	Fault	Yes	Stack operations and SS register loads.
13	#GP	General Protection	Fault	Yes	Any memory reference and other protection checks.
14	#PF	Page Fault	Fault	Yes	Any memory reference.

Tipos de Interrupciones

Table 6-1. Protected-Mode Exceptions and Interrupts (Contd.)

Vector	Mnemonic	Description	Type	Error Code	Source
15	—	(Intel reserved. Do not use.)		No	
16	#MF	x87 FPU Floating-Point Error (Math Fault)	Fault	No	x87 FPU floating-point or WAIT/FWAIT instruction.
17	#AC	Alignment Check	Fault	Yes (Zero)	Any data reference in memory.[2]
18	#MC	Machine Check	Abort	No	Error codes (if any) and source are model dependent.[3]
19	#XM	SIMD Floating-Point Exception	Fault	No	SSE/SSE2/SSE3 floating-point instructions[4]
20	#VE	Virtualization Exception	Fault	No	EPT violations[5]
21	#CP	Control Protection Exception	Fault	Yes	RET, IRET, RSTORSSP, and SETSSBSY instructions can generate this exception. When CET indirect branch tracking is enabled, this exception can be generated due to a missing ENDBRANCH instruction at target of an indirect call or jump.
22-31	—	Intel reserved. Do not use.			
32-255	—	User Defined (Non-reserved) Interrupts	Interrupt		External interrupt or INT *n* instruction.

Vamos a poder definir nuestras interrupciones en el rango 32 a 255.

Implementando interrupciones

¿Cómo se implementa el mecanismo asociado a cada tipo de interrupción?

- Escribimos la rutina de atención de cada excepción/interrupción, es importante utilizar `IRET` en lugar de `RET`.
- Definimos nuestra `IDT` con los descriptores correspondientes.
- Cargamos nuestro descriptor de `IDT` en `IDTR`.

Estos últimos dos puntos son similares a lo que hicimos para la `GDT`.

Estado de la pila

Antes de atender una excepción/interrupción el procesador pushea en la pila:

- El registro `EFLAGS`.
- El registro `CS`.
- El registro `EIP`.
- En algunas excepciones también un `error code`.

Por eso es importante utilizar `IRET` en lugar de `RET`, debemos sacar más cosas de la pila.

Estado de la pila

¿Si el `error code` es opcional, cómo sabe el procesador si tiene que quitarlo de la pila o no?

Respuesta:

No lo sabe, cuando escribamos el código de una excepción tenemos que consultar si la excepción pushea un código de error o no, y si lo hace, antes de llamar a `IRET` tenemos que quitarlo de la pila.

En la práctica de hoy no necesariamente ejercitaremos esto pero es importante que lo tengan en cuenta.

Estado de la pila

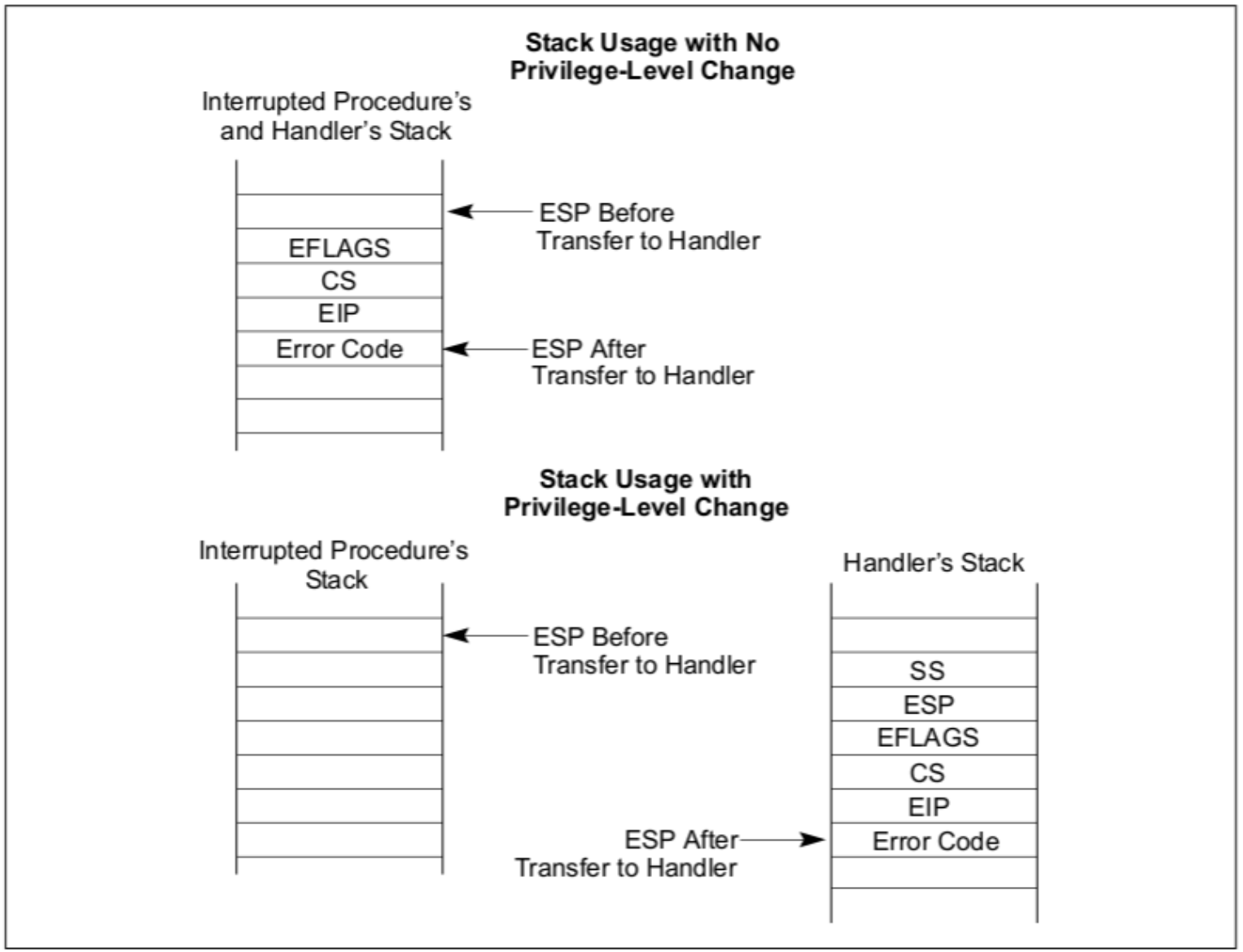


Figure 6-4. Stack Usage on Transfers to Interrupt and Exception-Handling Routines

Estado de la pila

¿Tenemos que mantener la pila alineada a 16 bytes?

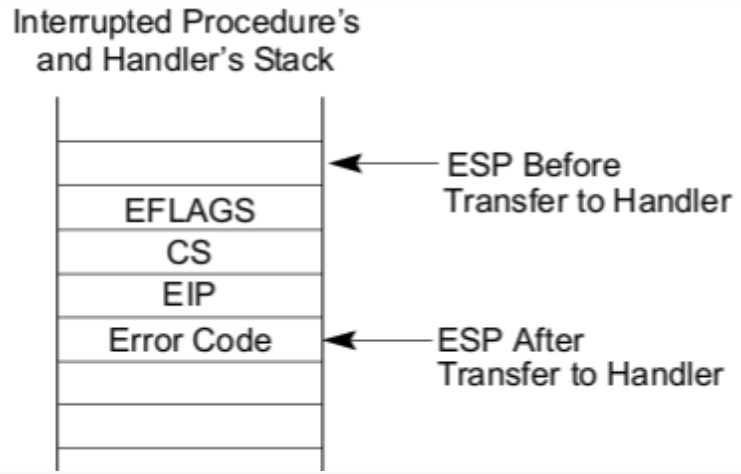
Respuesta:

No hace falta, recuerden que no estamos trabajando con `libc`, en general va a estar alineada a 4.

Estado de la pila

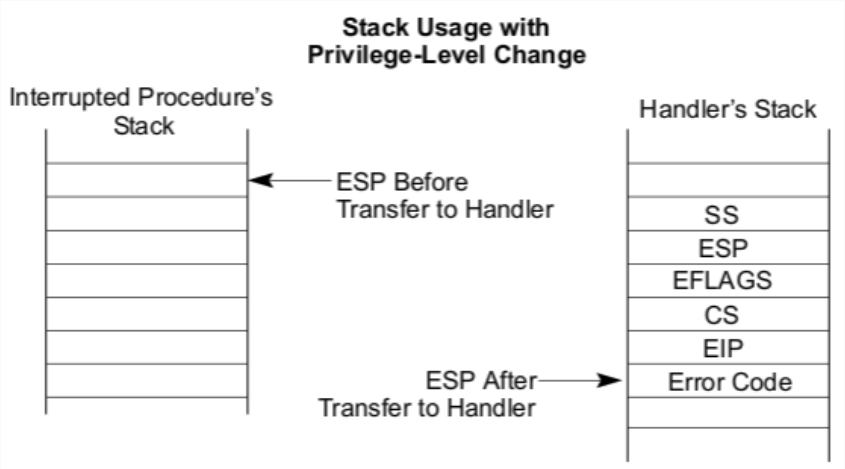
Por ahora no vamos a hablar de los cambios de nivel de privilegio, pero es importante saber que van a jugar un rol determinante en el manejo de tareas. Ahí veremos qué significa tener dos pilas.

Estado de la pila



Esta es la primera versión que nos interesa, entendiendo que el código de error sólo se pushea para algunas **excepciones** (no afecta interrupciones externas o internas). Es el estado de la pila cuando no hay cambio de nivel de privilegio (excepciones e interrupciones externas).

Estado de la pila



Esta es la segunda versión, entendiendo que el código de error sólo se pushea para algunas **excepciones** (no afecta interrupciones externas o internas). Es el estado de la pila cuando hay cambio de nivel de privilegio (en nuestro caso con interrupciones internas). Noten que estamos usando otra pila, cuando inicialicemos tareas veremos como se configuran estos valores.

Estructura de la IDT

Estructura general de la IDT

En términos estructurales, la `IDT` es una tabla muy parecida a la `GDT` donde cada entrada es un descriptor de compuerta (`gate descriptor`).
Al igual que con segmentación, para poder resolver las interrupciones el procesador va a emplear la `IDT` que se encuentre referida en el registro `IDTR`.

Compuertas de la IDT

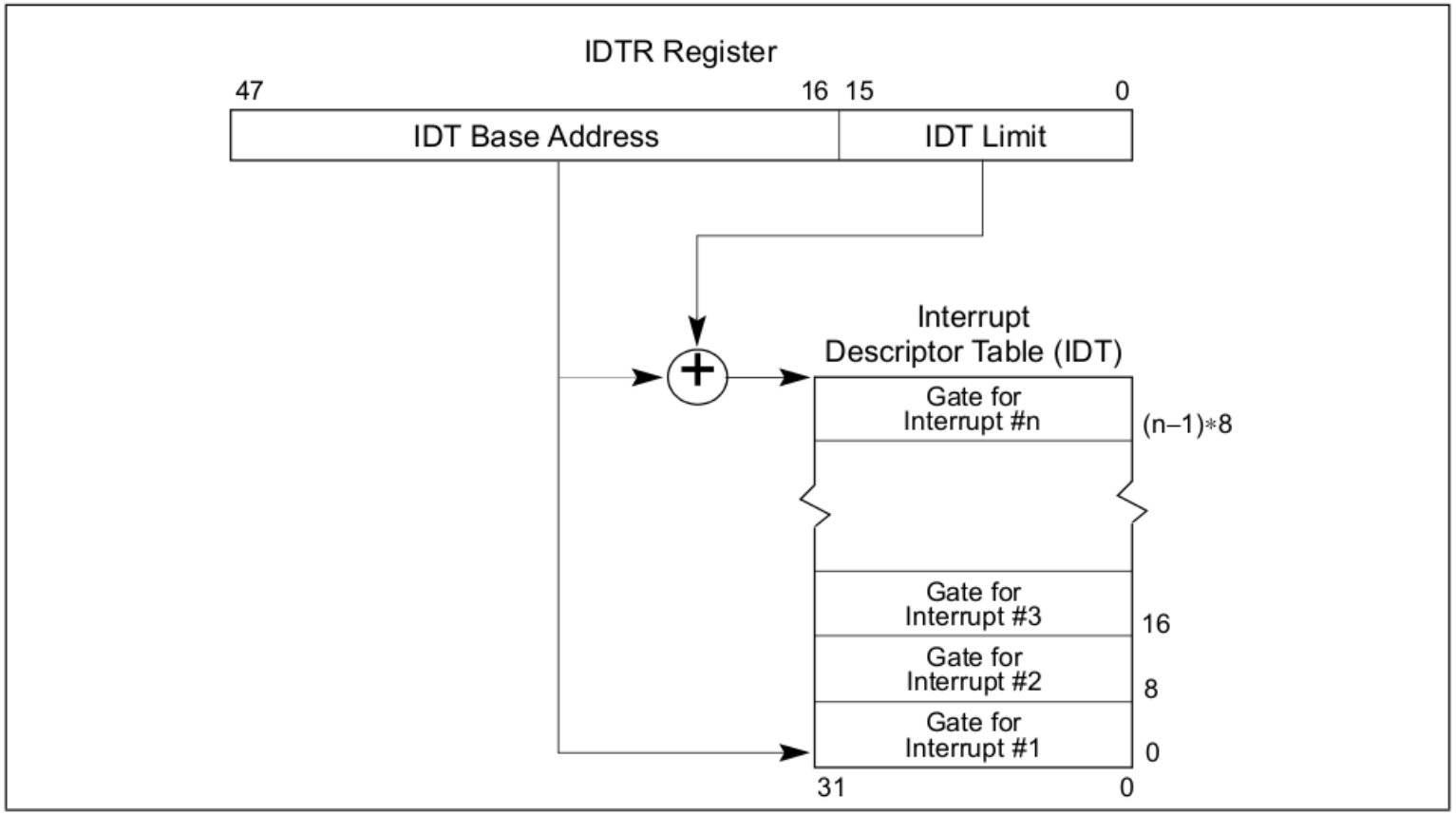


Figure 6-1. Relationship of the IDTR and IDT

Compuertas de la IDT

La `IDT` contiene descriptores para:

- **Compuertas de tarea** que estudiaremos cuando veamos tareas.
- **Compuertas de interrupción** que emplearemos en este taller.
- **Compuertas de trampa** que son similares a las de interrupción pero manejan distinto el estado del bit de interrupción en `EFLAGS` y no utilizaremos en la práctica.

Compuertas de la IDT

Así se ven los descriptores:

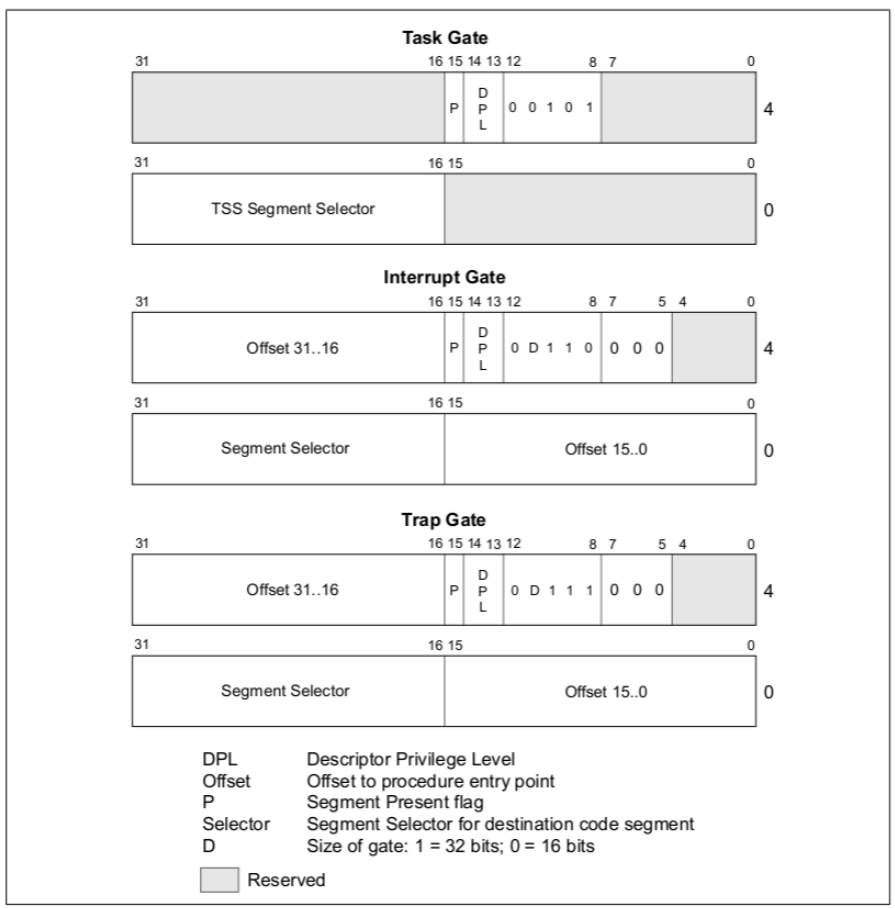
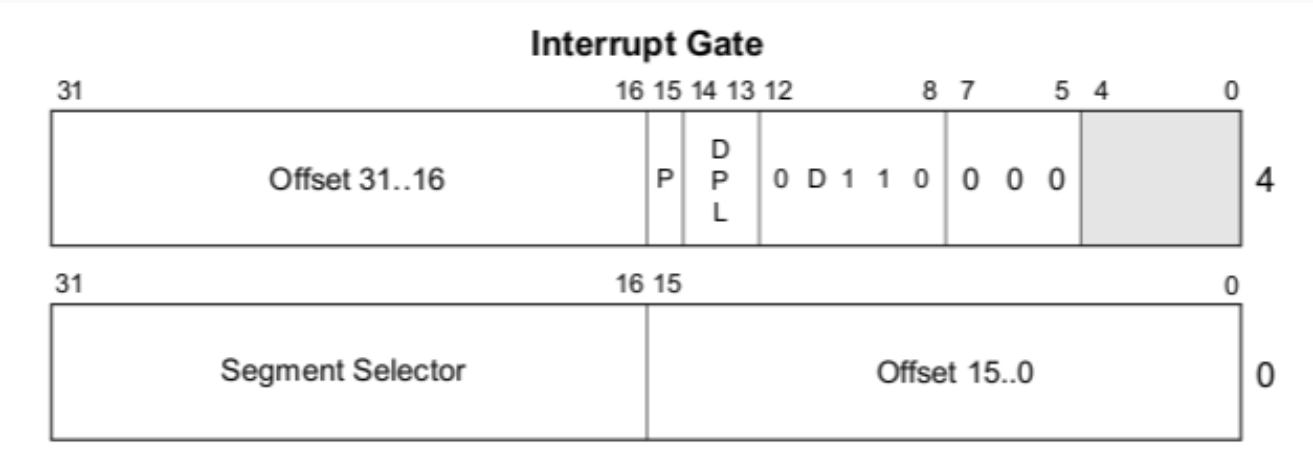


Figure 6-2. IDT Gate Descriptors

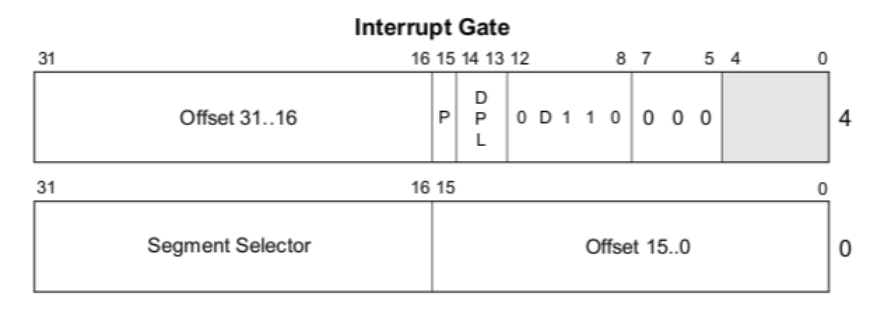
Compuertas de la IDT

Recuerden que en este taller utilizaremos compuertas de interrupción, esto quiere decir que nuestros descriptores o entradas de `IDT` se verán así:



Compuertas de la IDT

Veamos qué atributos son importantes:



- **Offset** que va a ser la dirección de memoria donde comienza la rutina de atención de interrupción.
- **Segment selector** que indica qué selector debe utilizarse al ejecutar el código de la rutina.
- **P,DPL** que indican si la rutina se encuentra en memoria o no y el nivel de privilegio, respectivamente.
- **Bits 8 a 12** de los bytes 4 a 7 indican el tipo específico de la compuerta de interrupción, el bit `D` indica si es una compuerta de 32 o 16 bits.

Atributo de tipo

Table 3-2. System-Segment and Gate-Descriptor Types

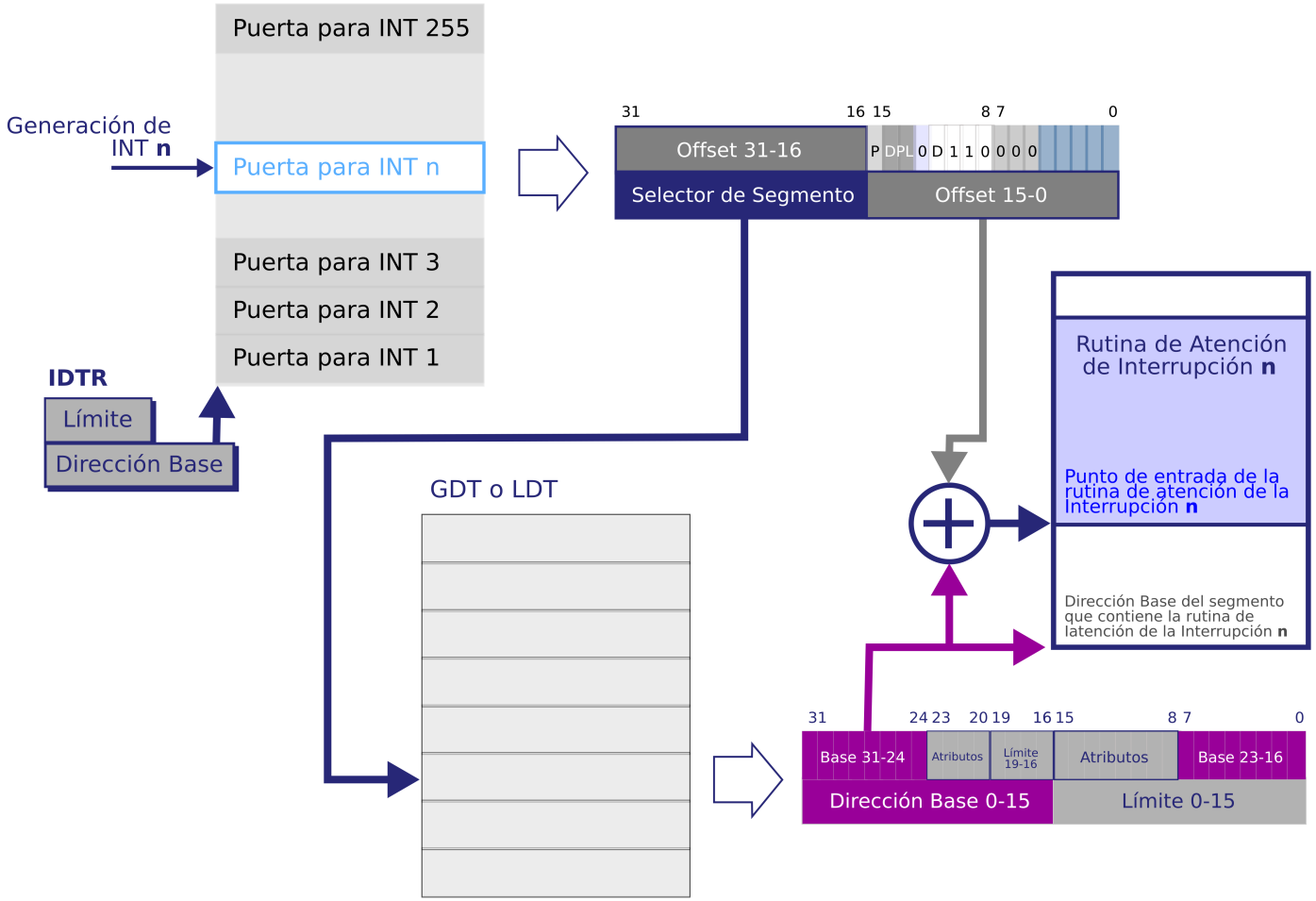
Type Field					Description	
Decimal	11	10	9	8	32-Bit Mode	IA-32e Mode
0	0	0	0	0	Reserved	Reserved
1	0	0	0	1	16-bit TSS (Available)	Reserved
2	0	0	1	0	LDT	LDT
3	0	0	1	1	16-bit TSS (Busy)	Reserved
4	0	1	0	0	16-bit Call Gate	Reserved
5	0	1	0	1	Task Gate	Reserved
6	0	1	1	0	16-bit Interrupt Gate	Reserved
7	0	1	1	1	16-bit Trap Gate	Reserved
8	1	0	0	0	Reserved	Reserved
9	1	0	0	1	32-bit TSS (Available)	64-bit TSS (Available)
10	1	0	1	0	Reserved	Reserved
11	1	0	1	1	32-bit TSS (Busy)	64-bit TSS (Busy)
12	1	1	0	0	32-bit Call Gate	64-bit Call Gate
13	1	1	0	1	Reserved	Reserved
14	1	1	1	0	32-bit Interrupt Gate	64-bit Interrupt Gate
15	1	1	1	1	32-bit Trap Gate	64-bit Trap Gate

El atributo de tipo determina qué representa nuestra entrada en la `IDT`, en este taller sólo usaremos el tipo `1 1 1 0`.

Esquema general

¿Qué mecanismos se ponen en juego para atender una interrupción?
Veamos cómo entran en juego tanto la `IDT` como la `GDT` para resolver la dirección de la rutina que responde al pedido de interrupción.

Esquema general



Archivos

1. `idt.h`

   Descripción de las estructuras

2. `idt.c`

   Estructura de la `IDT` con cada una de sus entradas

   - `idt_entry idt[255] = { ... };`
   - `IDT_ENTRY(numero)`

     Permite declarar una entrada en la `IDT`, para la utilizar el handler de nombre `_isrnumero`
   - `void init_idt()`

     Función llamada desde el kernel para inicializar las entradas en la `IDT`

Interrupciones de teclado y reloj

Interrupciones externas del taller

En este taller vamos a atender dos tipos de interrupciones externas, una de reloj y otra de teclado, veamos como implementar esto.

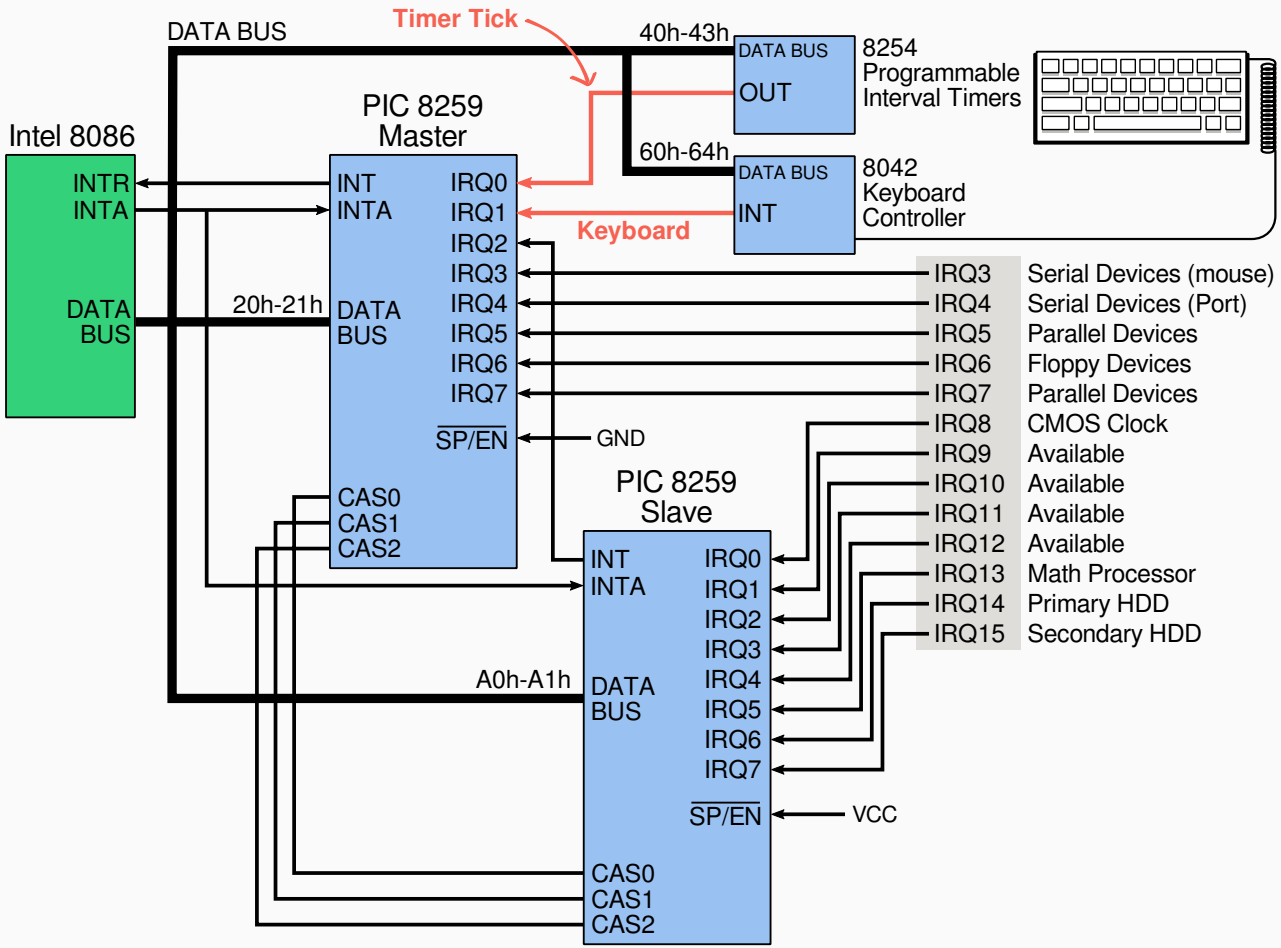
El controlador de interrupciones

- Recuerden que las interrupciones externas son administradas por un controlador de interrupciones (**PIC**).

- Es un elemento externo al procesador y el código necesario para su inicialización se provee **casi** resuelto al igual que lo hicimos con el `Bootloader`.

- Veamos cómo se ve el esquema de conexión entre el procesador, el PIC y los dispositivos.

Conexion de PIC 8259



Protected Mode Interrupt Table

INT 0 ⋮ INT 31	Excepciones del procesador	
INT 32	IRQ0	Timer Tick
INT 33	IRQ1	Keyboard
	IRQ2	Reserve
	IRQ3	Serial Devices (mouse)
	IRQ4	Serial Devices (Port)
	IRQ5	Parallel Devices
	IRQ6	Floppy Devices
	IRQ7	Parallel Devices
⋮	IRQ8	CMOS Clock
	IRQ9	Available
	IRQ10	Available
	IRQ11	Available
	IRQ12	Available
	IRQ13	Math Processor
	IRQ14	Primary HDD
INT 47	IRQ15	Secondary HDD
INT 48 ⋮ INT 255	Interrupciones de Software	

Configuración de PIC 8259

- Las interrupciones por hardware del procesador, **se pisan** con los indices seteados para los controladores de interrupciones.

- Se deben **remapear** los controladores a un espacio de interrupciones designado a dispositivos de entrada/salida.

- Para acceder a los `PIC` se usan las instrucciones "`in`" y "`out`".

- La direcciones `20h` y `21h` corresponden al `PIC1`, y las direcciones `A0h` y `A1h` al `PIC2`

Operaciones de entrada/salida

Las operaciones de entrada y salida utilizan **puertos**.

Podemos pensar en ellos como si fueran registros o posiciones de memoria que se identifican con un **número** (número de puerto) y un **tamaño** (8, 16, 32 bits por ej.).

El procesador lee y escribe de un puerto utilizando las operaciones `IN` y `OUT`. Podemos suponer que nos permiten leer y escribir en un registro particular de un dispositivo.

Configuración de PIC 8259 - Palabras de Control

La configuración se realiza enviando palabras en el siguiente orden, por los puertos indicados.

Address	Read/Write	Function
20h y A0h	Write	Initialization Command Word 1 (ICW1)
	Write	Operation Command Word 2 (OCW2)
	Write	Operation Command Word 3 (OCW3)
	Read	Interrupt Request Register (IRR)
	Read	In-Service Register (ISR)
21h y A1h	Write	Initialization Command Word 2 (ICW2)
	Write	Initialization Command Word 3 (ICW3)
	Write	Initialization Command Word 4 (ICW4)
	Read/Write	Interrupt Mask Register (IMR)

Rutinas del PIC

Los archivos `pic.h` y `pic.c` contienen las rutinas para controlar el PIC.

- **pic_reset**: Remapea los PICs a indices después de las excepciones del procesador.
- **pic_enable**: Habilita los PICs para que generen interrupciones.
- **pic_disable**: Deshabilita los PICs para impedir que generen interrupciones.
- **pic_finish1**: Indica al PIC1 que fue atendida una interrupción.
- **pic_finish2**: Indica al PIC2 que fue atendida una interrupción.

Rutinas del PIC

Código para activar interrupciones externas:

```
call pic_reset  ; remapear PIC
call pic_enable ; habilitar PIC
sti             ; habilitar interrupciones
```

Una vez activado el PIC debemos indicarle al procesador que debe responder interrupciones externas.

La instrucción **sti** setea el flag de interrupciones.

Configuración de PIC 8259 - Rutina de remapeo


```
; Inicializacion PIC1
mov al, 11h  ;ICW1: IRQs activas por flanco, Modo cascada, ICW4 Si.
out 20h, al
mov al, 8    ;ICW2: INT base para el PIC1 Tipo 8. ¿Queremos esto?
out 21h, al
mov al, 04h  ;ICW3: PIC1 Master, tiene un Slave conectado a IRQ2
out 21h ,al
mov al, 01h  ;ICW4: Modo No Buffered, Fin de Interrupcion Normal
out 21h, al  ;      Deshabilitamos las Interrupciones del PIC1
mov al, FFh  ;OCW1: Set o Clearel IMR
out 21h, al
```

Continuemos...

Configuración de PIC 8259 - Rutina de remapeo


```
; Inicializacion PIC2
mov al, 11h  ;ICW1: IRQs activas por flanco, Modo cascada, ICW4 Si.
out A0h, al
mov al, 70h  ;ICW2: INT base para el PIC2 Tipo 070h. ¿Queremos esto?
out A1h, al
mov al, 02h  ;ICW3: PIC2 Slave, IRQ2 es la lnea que enva al Master
out A1h, al
mov al, 01h  ;ICW4: Modo No Buffered, Fin de Interrupcion Normal
out A1h, al
mov al, FFh  ;OCW1: Set o Clearel IMR
out A1h, al
```

Ya tendríamos los dos PICs inicializados...

¡OJO con ambas `ICW2`!

Interrupción de reloj

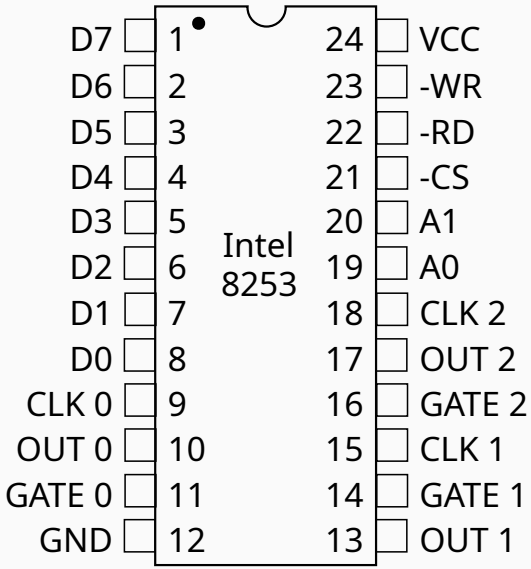
¿Que es la interrupción de reloj?

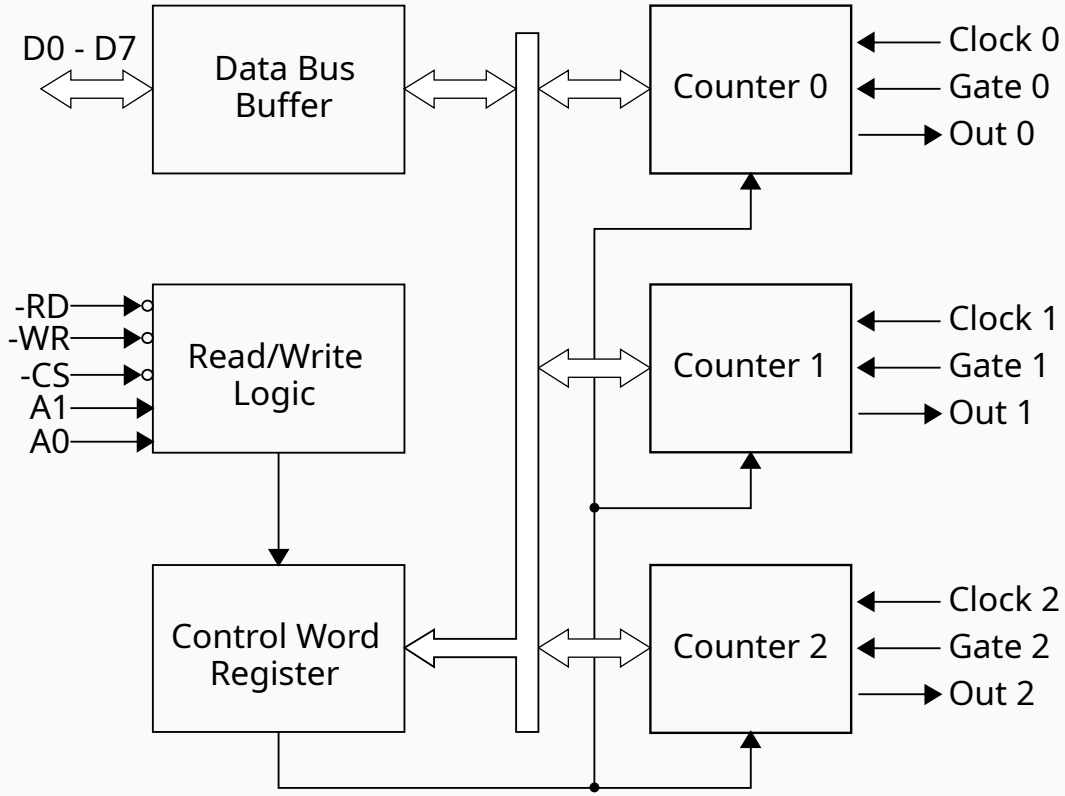
- Vamos a contar con un componente externo al procesador que va a generar una interrupción a intervalos regulares..

- Va a ser un elemento necesario a la hora de implementar el manejo de tareas, ya que utilizaremos esta interrupción para conmutar entre la tarea actual y la siguiente (**scheduling**).

Programmable Interval Timer (PIT) "El clock"

El integrado 8253/8254 tiene 3 contadores independientes. El registro de control (Control Word Register) permite programar cada uno de los tres contadores independientemente. Su tarea es **generar interrupciones** a intervalos regulares de tiempo (temporizador).





Keyboard controller

El integrado 8042 o controlador PS/2 permite controlar tanto el teclado como el mouse.

Uso:

. Leemos teclas a través del puerto **0x60** (`in al, 0x60`)

. Obtenemos un **scan code**

Scan code: Único para cada tecla

El teclado genera dos interrupciones:

Scancode=`0xxxxxxx` - **make**: Cuando se está presionando una tecla

Scancode=`1xxxxxxx` - **break**: Cuando se está soltando una tecla

Keyboard controller

Ejemplo:

Presionar **a** $\rightarrow$ scancode `0x1E`

Presionar **b** $\rightarrow$ scancode `0x30`

Soltar **a** $\rightarrow$ scancode **0x9E** (`0x1E + 0x80`)

Scan codes:

`http://www.win.tue.nl/~aeb/linux/kbd/scancodes-1.html`

(sección: "**1.4 Ordinary scancodes**").

Rutinas de interrupciones

Una vez activadas, nos comenzarán a llegar interrupciones del reloj (**32**) y del teclado (**33**).

Código para atender una interrupción externa

```
_isrXX:
        pushad
        ...
        call pic_finish1
        ...
        popad
        reti
```

Cuando atendemos una interrupción generada por el PIC.
Debemos notificarle que ya la atendimos. Para eso usamos rutina **pic_finish1**

Llamadas al sistema (syscalls)

Repaso de syscalls

. Recordemos que una llamada a sistema o `syscall`, es una forma que tiene el sistema operativo de exponer funcionalidad a las aplicaciones o espacio de usuarix.

. Pueden tener que ver con el manejo de procesos, sistemas de archivo, dispositivos de entrada/salida y otras cosas más.

. Hay varias formas de implementarlas pero la que empleamos en este taller son las **interrupciones de software**, o sea asociando una o varias syscalls a una interrupción particular. Una aplicación puede acceder a la syscall **76** haciendo: `int 0x4C`.

Implementación de una SYSCALL

La implementación de una syscall en nuestro caso se va a hacer con una rutina de atención, tal como lo hicimos para las excepciones o interrupciones externas:

```
global _isrXX
...
_isrXX:
        pushad
        ...
        popad
        iret
```

Esta rutina atiende a la interrupción `XX`, con un prólogo y un epílogo muy generales. No hay una llamada a `pic_finish1` ya que es interna y el PIC no tuvo intervención, y un retorno de función con `IRET`.

Repaso

Repaso de la introducción de hoy

Entonces, para poder utilizar interrupciones en nuestro sistema debemos:

- Definir una `IDT` con sus descriptores asociados para excepciones, interrupciones externas (reloj y teclado) y `sys_calls`.
- Escribir nuestras rutinas de atención de interrupciones con las consideraciones del caso, uso de `IRET`, llamar a `pic_finish1` si es interrupción externa, comunicarnos con el puerto correspondiente usando `in`, `out` si hace falta.
- Cargar `IDTR` y habilitar interrupciones.

Consultas y ejercitación

Consultas y ejercitación